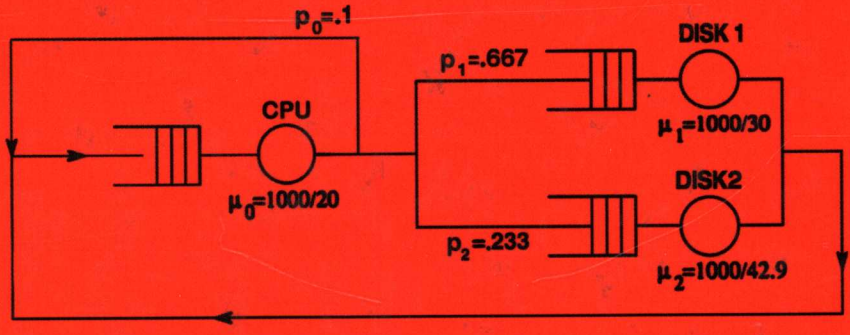
PERFORMANCE AND RELIABILITY ANALYSIS OF COMPUTER SYSTEMS

An Example-Based Approach Using the SHARPE Software Package



Robin A. Sahner
Kishor S. Trivedi
Antonio Puliafito

Kluwer Academic Publishers

PERFORMANCE AND RELIABILITY ANALYSIS OF COMPUTER SYSTEMS

An Example-Based Approach Using the SHARPE Software Package

Robin SAHNER
Urbana, IL

Kishor S. TRIVEDI
Duke University
Durham, N.C.

Antonio PULIAFITO
Jniversity of Catania
Catania, Italy

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

Sahner, Robin 1953-
Performance and reliability analysis of computer systems : an example-based approach using the SHARPE software package / Robin Sahner, Kishor S. Trivedi, Antonio Puliafito.
p. cm.
Includes bibliographical references and index.
ISBN 0-7923-9650-2
1. Electronic digital computers--Evaluation. 2. SHARPE.
I. Trivedi, Kishor Shridharbhai, 1946- . II. Puliafito, Antonio, 1965- . III. Title.
QA76.9.E94.S23 1996
004.2'4'01135133--dc20 95-38798
CIP

Printed on acid-free paper.

Printed in Great Britain by IBT Global, London

This printing is a digital duplication of the original edition.

PERFORMANCE AND RELIABILITY ANALYSIS OF COMPUTER SYSTEMS

An Example-Based Approach Using the SHARPE Software Package

PREFACE

Assessment of performance, reliability and availability is a key step in the design, analysis and tuning of computer systems. Suppose we have a multiprocessor system and we want to be sure it provides enough processing power. If we add a processor, how much better will performance get? Could additional overhead make the performance worse? Could we get a performance improvement just by changing the scheduling of jobs? How would adding a processor affect the reliability of the system. Would this make the system go down more often? If so, would an increase in performance outweigh the decrease in reliability?

Determining what questions to ask and what measures will address them involves examining the goals and requirements set out by system users. Performance requirements might be focused more on system throughput, on response time, or on meeting deadlines. That is, is it more important that a certain number of jobs or transactions can be processed per unit time, or that individual jobs can expect a certain average response time, or that all jobs are guaranteed a certain maximum response time? Reliability and availability requirements might be focused on measures like average system downtime, the likelihood that a system will stay up for a given amount of time or the mean time to system failure.

The relative importance of performance and reliability requirements will differ depending on the system environment and typical usage. Sometimes performance and reliability issues can be addressed separately, but sometimes their interaction demands a measure that combines aspects of both. For example, it might be required that once started, a system must get a certain amount of work before it goes down, but it does not matter how fast the work gets done or how long the system stays up.

Having decided what measures are needed, a system designer has several options when it comes to predicting their values:

- Make an educated guess based on experience with previous, similar systems.

- Build one or more systems (or prototypes) and take measurements.
- Use discrete-event simulation to model the system.
- Construct analytic models of the system.

These options are not exclusive; a system designer may very well use two or more of them, depending on the stage of the design process, the nature and rigidity of the system requirements, and the time and resources available. Each option brings something to the design process.

Discrete-event simulation and analytic models both allow designers to predict system behavior without having to build and measure a system. A discrete-event simulation is essentially a program whose execution mimics the dynamic behavior of the modeled system and provides measures of the behavior. An analytic model is essentially a set of formulas or equations describing the system; manipulating or solving the equations leads to results that describe the system behavior. In simple cases, equations can be solved to get a closed-form answer but more often a numerical solution of the equations needs to be carried out.

Discrete-event simulation models can capture system behavior in however much detail the modeler desires. Their drawback is that they can take quite a long time to run. Since the use of models typically involves changing the parameters many times, this can be a real concern. Analytic models are generally more of an abstraction of the system than a discrete-event simulation model, which means that the results might not be good predictors of system behavior. Modelers must be very careful to choose good abstractions, and take care in parameterizing the models and validating them. But once an analytic model is set up it is easy and fast to carry out trade-off studies, answer "what if" questions, perform sensitivity analyses and compare design alternatives.

There are circumstances when analytic models can provide information not easily obtained by any other method. It might be too expensive or time-consuming to build even one actual system unless we were sure it was going to meet the system requirements. It is also sometimes impossible to assure ourselves by measurement that a system satisfies the design criteria. This could be the case if we were trying to build a system so reliable that waiting for it to fail enough times to accurately estimate its reliability would take years. But, if we knew the reliability characteristics of the components and had a model that captured the structural relationships between the components, we make a mathematical prediction of the system reliability.

A system designer has a wide range of kinds of analytical models to choose from. Each model has its strengths and weaknesses in terms of accessibility, ease of construction, efficiency and accuracy of solution algorithms, and availability of software tools. No single kind of model is best, or even necessarily appropriate, for every system and every measure of interest.

Reliability models like fault trees are straightforward and easy to understand, and solution methods have been studied extensively. But they cannot easily represent non-independent behavior of components and are not easily generalized to incorporate performance considerations. Markov chains provide great flexibility for modeling reliability, performance, and combined reliability and performance (performability) measures. But they are not always intuitive and the size of their state space grows much faster than the number of system components, making model specification and analysis difficult. Queueing networks are extremely intuitive, and those that have product-form have efficient solution methods, but they cannot represent systems where there is simultaneous possession of resources.

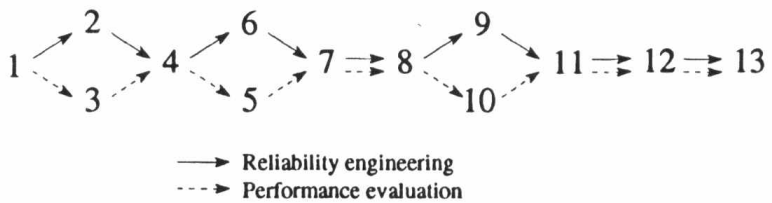
A modeler who is familiar with many different kinds of models, can easily choose models that best suit a particular system and the kind of measure that is needed at each stage of the design. It is also possible to use different kinds of models hierarchically for different physical or abstract levels of the system and to use different kinds of models to validate each other's results.

The purpose of this book is to provide a theoretical summary and examples of a variety of probabilistic, discrete-state models typically used to assess the reliability and performance of computer and communication systems. The models we have included are:

- combinatorial reliability models: reliability block diagrams, fault trees and reliability graphs;
- directed, acyclic task precedence graphs;
- Markov and semi-Markov models, including Markov reward models;
- product-form queueing networks;
- generalized stochastic Petri nets.

This book is suitable for three different types of users. It could be used as a text for a senior undergraduate or first year graduate course in several possible

departments: electrical engineering, computer science, industrial engineering, or applied mathematics. The following graph shows the chapters that could be covered for courses in reliability engineering and performance evaluation.



The book could be used as a supplementary text in a course on fault-tolerant computing or a course on computer architecture. It is also suitable for self-study by practicing engineers and can be used by researchers who need to know about models of system performance and reliability. A two semester sequence in calculus and an introduction to probability are sufficient background for most of the material in this book. Some of the material related to Markov chain analysis requires basic knowledge about linear algebra, differential equations and Laplace transforms.

Access to the software tool SHARPE, (Symbolic Hierarchical Automated Reliability and Performance Evaluator), will be extremely useful, if not essential. The book should be especially useful to current and potential users of SHARPE. To obtain a copy of the software, please contact the second author, Kishor Trivedi.

There are many other software tools that provide support for model specification and analysis [45, 48]. Most of the available tools support a single model type and many of them are tailored to a specific application domain or provide only limited user choice when it comes to the model characteristics and parameters.

We believe that SHARPE is a useful modeler's "toolchest" because it contains support for multiple model types and provides flexible mechanisms for combining results so that models can be used in hierarchical combinations. SHARPE allows its users to construct and analyze performance, reliability, availability and performability models. It gives users direct and complete access to the models without making any assumptions about an application domain.

This book is divided into two parts, with theory in Part I and examples in Part II. Part I begins in Chapter 1 by reviewing random variables and their distribution functions. Chapter 2 concentrates on reliability and availability modeling. It introduces formal definitions of reliability and availability, then describes reliability block diagram, fault tree and reliability graph models with worked-out examples of each. Chapter 3 turns to performance modeling, presenting a series-parallel, directed acyclic graph model. Chapter 4 discusses stochastic processes in general, and Markov chains in particular, with examples showing how these can be used to model either reliability or performance. Chapter 5 returns to performance modeling with a discussion and examples of product-form queueing networks. In Chapter 6, Markov reward models are presented as a way to combine performance and reliability measures into a performability measure. Chapter 7 discusses stochastic Petri net models. Chapter 8 presents semi-Markov models.

Part II consists of a large number of model examples. Chapter 9 presents reliability models, including reliability block diagrams, fault trees, reliability graphs, Markov chains, and stochastic Petri nets. Chapter 10 presents performance models, including acyclic series-parallel graphs, Markov chains, queueing networks, and stochastic Petri nets. In Chapter 11, we show how hierarchical, sometimes heterogeneous models can be used to model a system that does not lend itself to analysis by a single model. Chapter 12 contains Markov reward model examples. In Chapter 13, we discuss numerical problems that come up during model analysis and ways of dealing with or avoiding them.

All of the examples in Part II are analyzed using SHARPE. Each feature of SHARPE is explained the first time it is used and sometimes again in later examples, because we expect that readers may not look at the examples in order. Appendix B contains the complete SHARPE language description; you can look there if you read the examples out of order and need to know what a particular construct means.

There is more than one way to read this book. It can be read straight through. Some readers might want to alternate between Parts I and II; it makes sense to follow the reading of a chapter in Part I that describes a particular kind of model with the chapter or sections in Part II that contains examples of that kind of model. It would also be reasonable for some people to start reading with Part II and refer back to Part I as needed for background material on the model types.

The authors thank Phil Chimento, Varsha Mainkar, Jogesh Muppala, Herve Tardif, Roger Smith and Malathi Veeraraghavan for contributing to the devel-

opment of the SHARPE program. We also thank Ashutosh Aggarwal, Gianfranco Ciardo, Sachin Garg, Dimitris Logothetis, Steve Hunter, Ajay Kshemkalyani, Varsha Mainkar, Jogesh Muppala, Bruce Reznick, Andrew Rindos, W. Earl Smith, and Steve Woolet for reading and commenting on various drafts.

CONTENTS

PART I

MODELING THEORY

Part I focuses on the theory of analytical modeling. We present the basic concepts and terminology for each modeling technique, explain how the models are analyzed, and give examples.

To help compare and contrast the model types, the examples will include a running example in which the various modeling techniques are used to model different facets of the same system. The system for this running example consists of a fault-tolerant, multiprocessor computer with multiple memory modules. We assume that the system is able to detect a processor or memory module failure and reconfigure itself to continue operation without the failed component. Two system design alternatives will be considered, one where all memory modules are shared by all processors, and one where each processor has a private memory module and there are also shared memory modules.

- In Section 2.3, we use a reliability block diagram to model system reliability for the all-shared-memory-module case.
- In Section 2.4, we present fault tree models for both design alternatives.
- In Section 2.5, we validate the fault tree model for the second design with an equivalent reliability graph model.
- In Section 3.2, we turn from reliability to performance modeling and use an acyclic graph model to analyze the execution time of a parallel program running on systems using both design alternatives.
- In Section 4.8.2, we add the assumption that components that have failed can be repaired and use Markov models to analyze system availability.
- In Section 5.3.1, we show how queueing models can be used to analyze system performance for both design alternatives.
- In Section 6.2, we present Markov reward models for the performability analysis of the system.
- In Section 7.5, we show how a stochastic Petri net model can be used to validate and extend one of the Markov models from Section 4.8.2.